Towards Simulator-like Observability for FPGAs: A Virtual Overlay Network for Trace-Buffers

Eddie Hung
Dept. of Electrical and Computer Engineering
University of British Columbia
Vancouver, B.C., Canada
eddieh@ece.ubc.ca

Steven J. E. Wilton
Dept. of Electrical and Computer Engineering
University of British Columbia
Vancouver, B.C., Canada
stevew@ece.ubc.ca

ABSTRACT

The rising complexity of verification has led to an increase in the use of FPGA prototyping, which can run at significantly higher operating frequencies and achieve much higher coverage than logic simulations. However, a key challenge is observability into these devices, which can be solved by embedding trace-buffers to record on-chip signal values. Rather than connecting a predetermined subset of circuits signals to dedicated trace-buffer inputs at compile-time, in this work we propose that a virtual overlay network is built to multiplex all on-chip signals to all on-chip trace-buffers. Subsequently, at debug-time, the designer can choose a signal subset for observation. To minimize its overhead, we build this network out of unused routing multiplexers, and by using optimal bipartite graph matching techniques, we show that any subset of on-chip signals can be connected to 80–90% of the maximum trace-buffer capacity in less than 50 seconds.

Categories and Subject Descriptors

B.7 [**Integrated Circuits**]: Design Aids—*Verification*

Keywords

FPGA Debug; FPGA Prototyping; Verification; Trace-Buffers; Overlay Network

1. INTRODUCTION

As the achievable capacities of digital integrated circuits grow, the verification and debugging tasks are becoming increasingly difficult. A Mentor Graphics study found that whilst silicon density doubles every 18 months, designer productivity only doubled every 39 months, and that half of all designer effort was spent performing functional verification [5]. Designers make extensive use of simulation to verify that their designs operate as expected and to hunt for the cause of incorrect behaviour, however, simulation is slow; IBM engineers reported that software simulation was only able to reach 10Hz while their custom ASIC had a target frequency of 1.6GHz — a difference of over 8 orders in magnitude [2].

A growing number of designers are now opting to prototype their design using one or more Field-Programmable Gate Arrays (FPGAs). The same Mentor study found that 55% of industry employed FPGA prototyping techniques in 2010, an increase from 41% in 2007 [5]. FPGA prototyping enables significantly higher verification coverage compared to simulation, allowing designers to exercise their design using realistic scenarios (e.g. booting an operating system).

The primary challenge during FPGA prototyping is one of visibility. Unlike software simulation, in which the designer can view the behaviour of any signal in the design at any time step, in prototyping only those signals which drive output pins can be observed. This significantly limits debug productivity, since it is often difficult to deduce internal behaviour by only observing output signals. Providing *simulator-like visibility* to an FPGA platform is seen as one of the key technologies required as FPGAs scale to larger and larger capacities [18]. This paper is a step in this direction.

A common technique for increasing visibility is to insert trace-buffers into a circuit, and use these trace-buffers to record a history of a subset of internal signals during normal device operation. Altera and Xilinx provide tools enabling this [24, 1] and third-party solutions are also available [19, 17]. A key constraint of this method, however, is that the signals that a designer wishes to observe must be predetermined at compile-time, before the circuit is operational, and often, before the exact nature of the bug is known. As a result, a designer wishing to change the set of signals that are recorded would have to recompile his or her design. Recompiling and/or reconfiguring the design to observe different signals is referred to as a *turn* in [22]; often many turns are required during debug to narrow down the cause of unexpected behaviour. References [8, 6, 1] all show how incremental routing can be used to connect signals to trace-buffers or output pins without a complete recompile, however, these techniques are still slow (in [8], a re-route time of 2,000 seconds for a 100,000 LUT circuit is reported).

In this paper, we propose a method which accelerates the debug process by significantly reducing the amount of time required to perform a turn. We do this by allowing the designer to change which signals are to be connected to the trace-buffer without recompiling the design, and without requiring a re-route of signals between debug iterations. We achieve this by, at compile-time, embedding a flexible overlay network which multiplexes almost all combinational and sequential signals of the gate-level circuit into these trace-buffers. Unlike [19], the network is not built using

FPGA'13, February 11–13, 2013, Monterey, California, USA.

the normal soft FPGA logic. Instead, we *reclaim unused routing multiplexers within the FPGA fabric* and use them to implement this network. As a result, the area overhead due to this network is essentially zero. At debug time, we then configure this network by setting a small number of routing bits to connect selected signals to the trace-buffers. Using our technique, we can forward any signal selection of a designer's choosing to 80–90% of the on-chip trace capacity.

Although our approach falls short of a software simulator in that we can only observe a limited number of signals and for a limited number of clock cycles as constrained by trace-buffer capacity, crucially, we will show that our technique allows the designer to defer the selection of which signals to observe to debug-time. This negates the need to recompile the circuit whenever the signal selection is changed, greatly accelerating debug productivity.

2. RELATED WORK

On-chip observability can be enhanced using either scan- or trace-based techniques. Scan-based techniques involve connecting internal flip-flops sequentially; in FPGAs, this can be achieved using general-purpose soft-logic as in [21], where the area and delay costs can be prohibitive, or through dedicated device readback support [9]. Scan techniques can provide complete visibility into the state of all flip-flops in the design, but typically require that the circuit is halted before scan-out. This can greatly slow down their use for real-time debugging; reference [9] reported that viewing one flip-flop using device readback can take between 2 to 8 seconds.

Trace-based techniques operate by utilizing a portion of the FPGA's embedded memory resources to record a small subset of internal circuit values during continued device operation. Examples of trace IP offerings include Xilinx ChipScope Pro, Altera SignalTap II and Synopsys Identify [24, 1, 17]. For these products, the subset of signals that are connected to trace-buffers must be determined by the designer ahead of time, before the circuit is compiled. Once trace instrumentation has been inserted, if a designer wishes to modify the observed signals, the circuit would often need to be recompiled. To combat this, commercial tools support general purpose incremental-compilation techniques whilst researchers have also proposed a trace-specific procedure in [8].

Most similar to our work is Tektronix Certus [19], which allows designers to specify a large subset of signals to connect to a proprietary Observation Network during compilation, from which they can select a smaller subset to trace at runtime. A related product is Altera SignalProbe [1], which uses incremental ECO techniques to multiplex up to 256 circuit signals to each reserved I/O pin for external analysis. Our work differs in that we do not require the designer to predetermine which signals they can observe — we aim to provide the designer with complete visibility, nor do we use general-purpose logic to provide this runtime flexibility.

Authors have also proposed exploiting an FPGA's unique reconfiguration capabilities for debug. Reference [7] describes a method to reclaim spare FPGA resources for debug by speculatively inserting trace-buffer logic connected to a set of "influential" signals without any user intervention. Signals used for this purpose are determined using automated selection techniques such as those presented in [11, 10]. In a proposal similar to our approach, Moctar et al. [13] reuse the local routing multiplexers present inside each FPGA logic cluster to implement the programmable shift operation

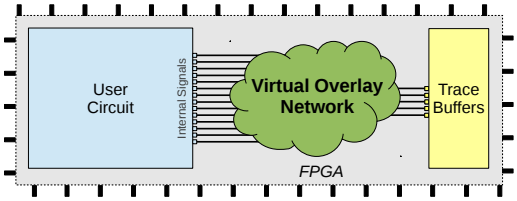


Figure 1: Virtual overlay network for multiplexing a large set of circuit signals to a small number of trace-buffer inputs

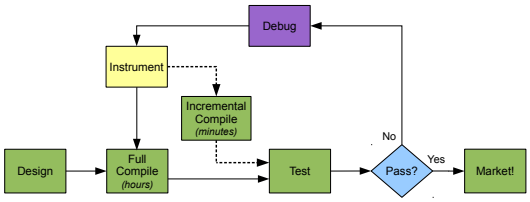
in floating-point computation, freeing up valuable soft-logic resources to be more efficiently used elsewhere.

Similar to [8], one important aspect that enables this proposed work is the realization that circuit signals can be connected to *any* free trace-buffer input for it to be observable. Instead of using a fully-populated crossbar where every network input can be forwarded to any network output, this flexibility can be exploited by using a sparse (n, m) concentrator network [14], which guarantees that any size-m combination of the size-n input set can be routed through to its outputs in an arbitrary order. However, because we intend to connect all circuit signals to our overlay network, and allow all of them to access as many trace-buffer inputs as possible, even a concentrator may add too much routing pressure to the FPGA. In this paper, we pursue a blocking network which sacrifices the absolute guarantee of any-m-of-n, but as the results later show, approximately 80–90% of full connectivity can still be achieved.

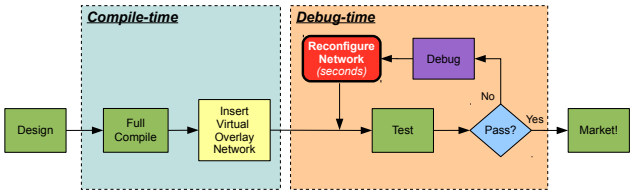
3. OUR APPROACH: OBSERVATION WITHOUT RECOMPILATION

We propose a method to allow the set of signals connected to on-chip trace-buffers to be modified without requiring the circuit to be recompiled. These techniques can be used to greatly reduce the time between debug turns, and hence rapidly accelerate the debugging flow. The key enabling component to this work is that, instead of building a custom FPGA mapping to connect each signal to one dedicated trace-buffer input as in existing IP [24, 1], we insert a virtual overlay network which allows multiple signals to be multiplexed to each trace-input; this is illustrated in Figure 1. Subsequently, changing the signals that are forwarded over this network will require only the virtual network to be reconfigured, rather than a new place-and-route solution.

Existing work pursue a debug flow similar to that shown in Figure 2a, in which the instrumentation procedure requires a new FPGA mapping to be constructed for each new set of observed signals at each debug turn. Whilst incremental compilation techniques can be used to accelerate this procedure [8], it still requires the entire circuit to be loaded into the memory of a CAD tool and some amount of additional routing (and perhaps placement) operations to be performed. Figure 2b describes our proposed debug flow. This debug flow consists of two phases: compile-time and debug-time. During compile-time, the uninstrumented circuit is fully compiled as normal, and the resulting mapping is then *completely* fixed. Next, the virtual overlay network is then added incrementally, using only the FPGA resources that were leftover from the initial mapping — this is described in Section 7. It is also possible to insert the overlay network during the original full compile, though this option is not explored here.

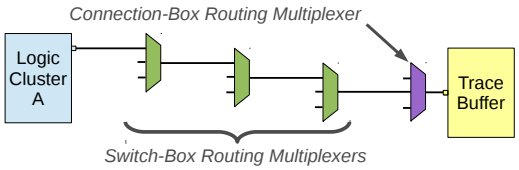


(a) Existing debug flow

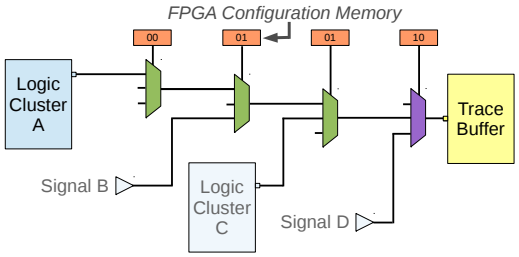


(b) Proposed debug flow: compile- and debug-time phases

Figure 2: Existing and proposed debug flows



(a) Prior work: single trace connections



(b) Proposed: overlay network with multiple connections

Figure 3: Trace-buffer connectivity

At debug-time, this overlay network is then repeatedly configured with the signals that a designer wishes to observe, and the device tested at each turn in order to record the desired signal values, until the root-cause of the bug is located. By eliminating all forms of recompilation from the inner loop, each debug turn can now be completed in a matter of seconds.

A number of key technical challenges have to be overcome in order to realize such a proposed flow, and these will be described as follows: Section 4 presents the details of our virtual overlay network, which seeks to connect all combinational and sequential signals of the user-circuit to the available trace-buffers. Section 5 describes the graph-based method we employ for computing a valid network configuration Lastly, Section 6 describes how this configuration can be programmed into the device.

4. VIRTUAL OVERLAY NETWORK

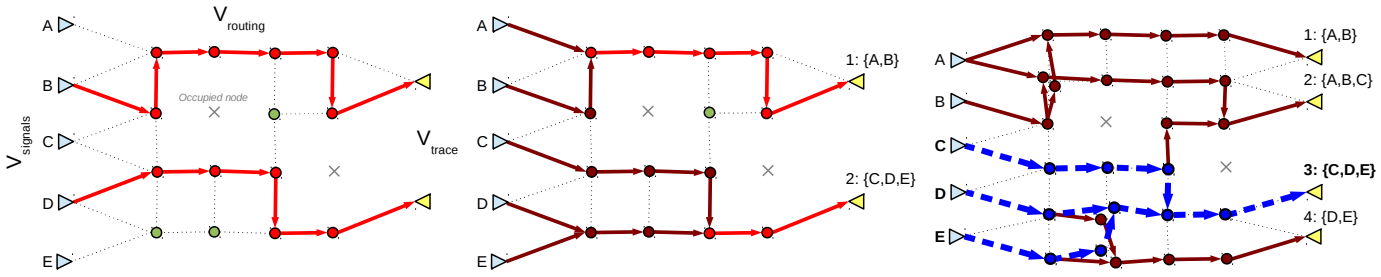
In this section, we will describe the details of our virtual overlay network. The key purpose of this network is to multiplex *all* on-chip signals to *all* trace-buffer inputs. With previous work, it is necessary to compile a custom point-to-point network for each new signal selection, generating a mapping such as that illustrated by Figure 3a where each observed signal is connected exclusively to a single trace-buffer input via a set of dedicated routing multiplexers. Instead, we propose that an overlay network is created out of these routing multiplexers, as shown in Fig. 3b, where a total of 4 signals: A, B, C and D are now connected to the same trace-input. The select-lines to each of these routing multiplexers are driven by the FPGA configuration memory — methods to reprogram their values are covered in a subsequent section. The reconfigurable nature of FPGAs arises from the abundance of multiplexers inside, and by utilizing routing multiplexers to build this overlay network instead of general-purpose user-logic, this network can be built much more efficiently. The feasibility of this proposal is supported by analysis that in mapping our set of uninstrumented benchmark circuits to a minimum array size FPGA with a small amount of routing slack, only 32–51% of the total interconnect capacity (geomean at 41%) was utilized.

We note that our approach of instrumenting the circuit after compilation means that only gate-level signals are accessible; but due to the effects of optimization or technology-mapping, such signals may not posses a direct one-to-one correspondence to those at the RTL or HDL-level signals that a designer is most familiar with. We believe several approaches exist to alleviate this mismatch: first, unless register re-timing is performed, both commercial and academic CAD tools preserve the names of all sequential signals in the design, which designers can use as fixed points of reference. With sufficient visibility into the sequential signals that affect it, any intermediate combinational signals can then be re-computed using offline simulation.

Second, designers are able to manually specify additional points of reference by using synthesis attributes to prevent combinational signals from being optimized away — the "`syn_keep`" attribute is available in Quartus II, and Synplify, whilst the "`S`" (SAVE_NET) attribute in ISE. Trace IP which instrument at the RTL or HDL-level (SignalTap II, ChipScope, Certus) implicitly do this. During prototyping, large circuits are likely to be I/O-bound due to the need to partition the circuit amongst multiple FPGAs — in those cases, it would be feasible to optimize the circuit less aggressively so that more combinational signals can be preserved. This approach is not dissimilar to debugging software applications, where speed is traded for visibility; in fact, the upcoming version of GCC 4.8 supports a new "`-Og`" optimization level to address this.

Building the virtual overlay network is essentially a routing problem. This routing problem can be represented as a routing resource graph $G(V, E)$. We define $V = V_{signals} \cup V_{routing} \cup V_{trace}$ where $V_{signals}$ is the set of all circuit signals that can be traced, $V_{routing}$ the set of unused routing multiplexers, and V_{trace} the set of trace-buffer inputs. E is the set of unused routing tracks that exist between these resources. Example routing resource graphs are shown in Figure 4 where $V_{signals}$ is indicated by $\triangleright$ triangles, $V_{routing}$ as $\circ$ circles, and V_{trace} as $\triangleleft$ triangles. Fig. 4a illustrates an example point-to-point network that would be created by prior work for observing signals B and D. Here, each routing multiplexer would be used to carry only one signal.

Figure 4b shows a routing solution for the same resource graph in which all five circuit signals are connected to either of



(a) Prior work: point-to-point connections (b) Our approach: utilize a union of trees, each rooted at a trace-pin (c) Proposed: Virtual overlay network (tree for trace-input 3 is highlighted)

Figure 4: Routing Resource Graphs $G(V, E)$: ▷ indicate circuit signals, ○ for routing multiplexers and ◁ are trace-buffer inputs

the two trace-buffer pins available. Each routing multiplexer can have a fan-in of more than one. At debug-time, a designer can now configure the routing multiplexers in such a way as to forward just one signal to each trace-input. For this particular solution, designers can observe any single signal in their circuit, and a limited selection of any two signals simultaneously, as defined by the Cartesian product of the two signal sets {A,B} × {C,D,E}: {AC,AD,AE,BC ...}.

The key feature of this routing solution is that it is made up of a disjoint union of trees, each rooted at a trace-buffer input, with the leaves of each tree being the circuit signals that it connects. We use a disjoint union of such trees, to allow signal selections to be made for each trace-buffer input *independently* of other trace inputs; it is this constraint which differentiates and abstracts our virtual overlay network from the more general routing problem faced when building point-to-point networks. Whilst each trace-buffer input in the general routing resource graph G can be considered the root of a much larger tree which touches all the signals in its fan-in cone, the union of such trees will not be disjoint and hence signals for each trace-input cannot be selected independently.

Our virtual overlay network can be described as a graph $G'(V', E')$ where V' now consists of $V_{signals} \cup V_{trace}$, and E' the set of edges that describe connectivity between a circuit signal and a trace-pin. Furthermore, rather than connecting each signal in the circuit to a trace-buffer input just once as in Fig. 4b, it is possible for a signal to be a leaf of multiple trees. A valid routing solution for a network where this is the case is shown in Figure 4c; here, by occupying a few more routing resources, each of the five signals can now be connected to two of the four trace-buffer inputs. The increased flexibility of this overlay network can now guarantee that any combination of two signals can be selected for observation, but in practice, many more signals are possible.

5. NETWORK MATCHING

So far, we have assumed that at debug-time, the designer chooses which signal they wish to connect to every input pin of every trace-buffer in their circuit. Once this decision is made, a simple algorithm can be used to determine the select bits for each of the $V_{routing}$ multiplexers that make up the overlay network. This algorithm follows a greedy strategy: starting at the leaf node of the desired V_{signal}, move through all $V_{routing}$ multiplexers in the routing resource graph G belonging to the signal tree towards its root, V_{trace}. At each $V_{routing}$ multiplexer encountered, set it to forward the output from the previous node. However, making the choice

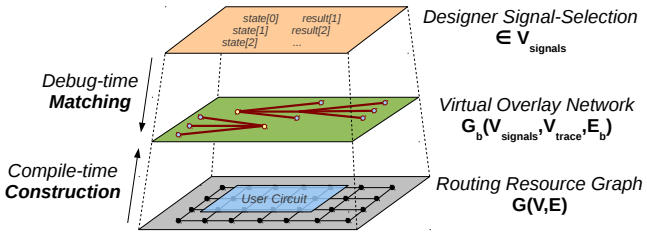
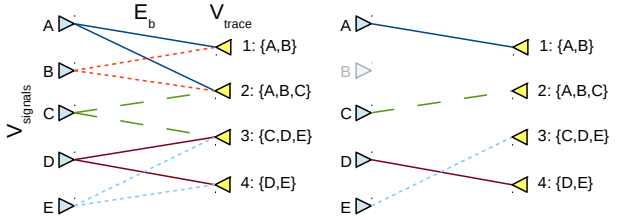


Figure 5: Virtual overlay network abstraction



(a) Original bipartite graph (b) Max. matching of graph

Figure 6: Bipartite graph $G_b(V_{signals}, V_{trace}, E_b)$ capturing signal/trace-input connectivity of virtual overlay network

of which signal to forward to which trace-input is not trivial. Consider the network in Figure 4c: although the designer can select any combination of two signals, they can only select a limited combination of three signals — defined by the Cartesian product of all sets. Given that each signal can be connected to one of two trace-buffer pins, there exists a problem of deciding which signals to connect to which pin. As an example, suppose that a designer wishes to observe the signals ACD; A can only be forwarded to trace-input 1 or 2, C to 2 or 3, whilst D can be forwarded to 3 or 4. From this list of constraints, a feasible assignment must be found: A→1, C→2, D→3 would be one valid solution, as would A→2, C→3, and D→4. However, assigning A→2 and D→3 would prevent signal C from reaching any trace-buffer input. Although this example was easy to compute by hand, it may not be so simple for circuits containing 10,000s of signals, connected to 1,000s of trace-inputs, from which 100s of signals are selected.

To solve this assignment problem, we utilize matching techniques for bipartite graphs. A bipartite graph can be described as $G_b(U_b, V_b, E_b)$, where U_b and V_b represent two disjoint sets of vertices, and E_b the set of edges that connect

between them. Edges must not exist between elements in the same set: from U_b to U_b, nor from V_b to V_b. The definition for our virtual overlay network fits this pattern, when substituting $U_b = V_{signals}$, the set of all circuit signals, $V_b = V_{trace}$, the set of all trace-buffer inputs, and $E_b = E'$ — the set of edges which describe the network connectivity between the two. The relationship between the virtual overlay network G_b and the general routing resource graph G is shown in Figure 5. The bipartite graph capturing the connectivity of the overlay network from Fig. 4c is shown in Figure 6a.

A maximum matching in a bipartite graph can be computed in polynomial time. A matching of graph G_b represents a subgraph of G_b in which none of its edges share a common vertex; a maximum matching is the largest such subgraph that can be formed. This is a very convenient property for computing which signal to forward to each trace-pin: given that each pin can only support one such connection, therefore, each node in V_{trace} must have at most one edge. The maximum matching solution for selecting signals ACDE from its bipartite graph is shown in Figure 6b, which returns the solution: $A \rightarrow 1$, $C \rightarrow 2$, $D \rightarrow 4$ and $E \rightarrow 3$. The maximum number of edges that can exist in a maximum matching is the minimum value of $|V_{signals}|$ and $|V_{trace}|$. In typical circuits, we would expect that more circuit signals exist than trace-inputs, and hence $|V_{signals}| >> |V_{trace}|$.

An additional useful characteristic of the maximum matching algorithm is that it not only returns a pass-fail result, for cases where a complete match is not possible, it will return a best-effort partial assignment. Because the virtual overlay network that we build is blocking, a maximum match can be used to return partial, but optimal, result where the maximum number of signals possible are forwarded over the network. In cases where not all requested signals can be forwarded, more than one maximum partial-match may exist — currently, only an arbitrary match is returned. Similarly, the solution may not capture designer intent in situations where higher emphasis is placed on certain signals — they may prefer one high-value signal to be selected over multiple, lower-value ones. This is a scenario we plan to address in future work using maximum weighted matching techniques.

6. NETWORK RECONFIGURATION

Once the select bits for each of the $V_{routing}$ multiplexers in the overlay network have been computed, the final task is to program these bits into the FPGA. For this we propose two different approaches: one which requires the FPGA to be powered down and fully reprogrammed, and an alternative which, with the correct architectural support, would allow the signal selection of a live FPGA to be changed on-the-fly.

6.1 Static Reconfiguration

The flow employed in existing work [24, 1, 8] is to create a new point-to-point circuit mapping for each new signal selection. The resulting bitstream would then be used to fully reprogram the all of the configuration memory on the FPGA device. This static reconfiguration procedure is identical to that which is undertaken during the initial power-on of the FPGA, and is also responsible for resetting all flip-flop and memory contents to a known value, destroying any existing user-state. For this reason, after reprogramming each new trace configuration, designers must then rerun their tests from scratch to collect their new signal trace.

In our proposed flow, because we do not recompile the circuit between each debug turn, we do not automatically generate a new bitstream. However, with exact knowledge of where the configuration bits for each routing multiplexer is located within this bitstream, it would be possible to directly modify only those bits necessary for configuring our overlay network. Then, when the FPGA device is statically reprogrammed, the desired signal selection is forwarded for observation. Graham et al. [6] adopt this bitstream modification approach for creating point-to-point trace networks.

6.2 Dynamic Reconfiguration

Alternatively, it may be possible for the overlay network to be changed for a new signal selection without losing user-state or interrupting live FPGA operation by using dynamic, partial reconfiguration. This feature allows circuit designers to dynamically reprogram only a portion of their FPGA during runtime, whilst the rest of the device continues functioning as normal.

Major FPGA vendors support dynamic reconfiguration in their high-end parts, and provide fine-grained, non-glitching support [25, 3] which does not corrupt user-state, showing the feasibility and viability of our application. Altera states that individual routing multiplexers in their fabric can be reconfigured; Vansteenkiste et al. [20] have also proposed that FPGA circuit-specialization be created using such fine-grained reconfiguration support.

Interestingly, current architectural support for reconfiguration goes beyond the needs of our transparent, observe-only trace-buffer network, as it enables all aspects of the FPGA to be reconfigured, including logic elements and lookup-tables, logic clusters, memory and DSP blocks, as well as their associated routing resources. Our network requires only the latter (specifically, only the configuration cells for all routing switch-boxes, as well as all connection-boxes for just the memory resources as shown in Fig 3b). Unfortunately, due to the proprietary nature of commercial FPGAs, we are unable to quantify what savings can be made here, nor to test our techniques on a physical device.

7. METHODOLOGY

To evaluate the feasibility of our virtual overlay network, we implemented our techniques using the FPGA CAD tool VPR, which forms part of the Verilog-To-Routing academic project [15]. Using VPR 6.0, we packed, placed and routed a set of benchmark circuits as normal onto the default VPR architecture (given in Table 2, but with an increased Fc_out of 0.2 as done in [8]) to generate the baseline data outlined in Table 1. In this flow, packing is performed with the objective of minimizing logic cluster usage, and placement is subsequently performed onto the minimum-sized FPGA array that will fit the circuit. The minimum channel width, W_{min}, is a measure for the routing efficiency of the CAD tools and FPGA architecture involved, and describes the absolute minimum number of routing tracks that is possible to implement the circuit on the given FPGA.

We make the same assumptions as in [8] that any free memory-block in the FPGA can be transformed into a trace-buffer for zero overhead, its contents can be extracted for free (using device readback techniques, or built in JTAG logic) and that triggering to control when to start and stop tracing is specified by the designer manually, or driven externally from a global pin. We do not believe these to be unrealistic

Circuit	6LUTs	Flip-Flops	FPGA Size	W_{min}	I/O	Logic Clusters	Multipliers	Memories
or1200	2963	691	25x25	72	**779/800**	298/475	1/18	2/12
mkDelayWorker32B	5580	2491	42x42	76	1064/1344	560/1302	0/50	**41/42**
stereovision1	10366	11789	43x43	70	278/1376	**1365/1376**	38/50	0/42
stereovision0	11462	13405	45x45	44	354/1485	**1479/1485**	0/66	0/42
LU8PEEng	21954	6630	59x59	86	216/1888	**2583/2596**	8/98	45/72
stereovision2	29849	18416	84x84	118	331/2688	3635/5208	**213/231**	0/154
bgm	30089	5362	69x69	88	289/2208	**3419/3519**	11/153	0/99
LU32PEEng	75530	20898	110x110	130	216/3520	**8861/9020**	32/378	150/252
mcml	99700	53736	119x119	86	69/3808	**10436/10591**	30/435	38/285

Table 1: Benchmark summary (values in bold indicate the limiting resource)

FPGA Architecture Parameter		Value
Logic Cluster Size	N	10
Lookup Table Size *(non-fracturable)*	K	6
Inputs per Cluster	I	33
Channel Segment Length	L	4
Cluster Input Flexibility	Fc_in	0.15
Cluster Output Flexibility *(default: 0.10)*	Fc_out	0.20

Table 2: FPGA architecture used, based on Altera Stratix IV

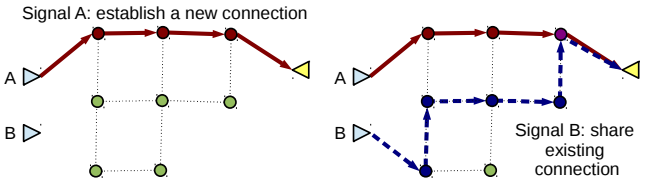


Figure 7: Circuit signals can either establish new trace connections (signal A) or share existing connections (signal B)

assumptions, given that the memory blocks inside the Xilinx Virtex family have built-in hard logic to implement FIFO functionality [23] with which we can build a ring-buffer to constantly record signal samples until halted by the trigger. In the FPGA architecture used, the widest configuration for each memory block is 72 bits (by 2048 entries) — this is adopted for our trace-buffers.

Rather than operating on each circuit at their minimum channel width, we inflate this value by a small amount, 30%, in order to reflect a realistic commercial architecture in which routing resources have been over-provisioned above the very best-case; this is a common approach also taken by other researchers such as [16]. To perform our experiments, we use our custom version of VPR to install our trace-buffer network incrementally — using only the spare resources not used in the original mapping — in a similar manner to [8]. However, instead of employing these techniques to build a custom point-to-point network, we have modified them to build our overlay network whilst preserving the guarantee that no existing circuit blocks nor routing are moved or re-routed. We then sweep each circuit to find the maximum number of times that all circuit signals can be connected to a different trace-buffer pin, a parameter we refer to as network connectivity. Once a feasible overlay network is found, we record the signals connected to each trace-pin into a text file which is subsequently used for matching at debug-time.

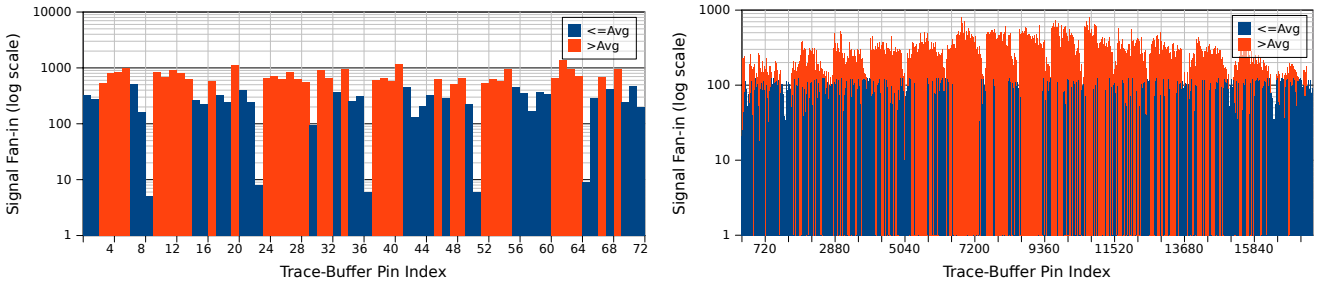
Currently, the VTR flow supports mapping circuits with only a single clock-domain. However, we believe that our approach can be extended to those with multiple clock-domains — given that each trace-buffer can only record signals from a single domain, we can also build an separate virtual overlay network to support all trace-buffers from each clock.

7.1 Compile-Time Construction

At compile-time, the virtual overlay network is constructed once per circuit. The primary challenge in constructing this overlay network using normal CAD tools is that these tools are designed to build a circuit mapping where each and every routing resource can, at most, be used once to connect one net source to one (or more) net sinks. The proposed network requires the reverse of this — we require multiple net sources to feed (multiplexed) a single trace-buffer sink.

Within VPR 6.0's routing stage, the PathFinder algorithm is employed to iteratively resolve routing resources that become overused, by slowly increasing their costs so that only the most critical nets can afford them, through a process known as negotiated congestion [12]. The goal of our algorithms is to attempt to connect all circuit signals — both combination and sequential — to the requested number of trace-buffer inputs, using a directed-search strategy which terminates whenever any input is found. However, rather than directing each net towards its nearest trace-buffer so that its routing wirelength, and hence any routing congestion, is minimized, we have found experimentally that higher network connectivity is possible if each circuit signal was directed towards a randomly chosen trace-buffer. We believe this is because it is beneficial to establish connections to trace-buffers that circuit signals would not normally prefer in order to fully utilize the flexibility provided by as many trace-pins as possible; this also has the added benefit that because signals are randomly distributed, higher quality signal to trace-pin matches, which are explained in the next section, can be achieved.

Instead of building point-to-point trace connections, where each routing resource can be used only once, we allow all $V_{routing}$ multiplexers to be overused, with the understanding that their select bits can be determined at debug-time. During network insertion, circuit signals have two options: either they can establish new connection to a new trace-input (signal A in Fig. 7), or they can branch onto an existing connection (signal B). A naïve approach would be to force all signals to always take the latter option whenever a used $V_{routing}$ node is encountered. However, we found that this made the solution sensitive to the order in which nets were routed: those processed first would be able to consume all the resources that suited itself most, with no regard for other nets, and hence causing all subsequent nets to work-around those connections. This is not desirable; we need to allow existing connections to be ripped up and relocated if it will lead to a globally better solution.



(a) `mkDelayWorker32B` with 5 connections per signal

(b) `mcml` with 21 connections per signal

Figure 8: Signal fan-in for overlay network — histogram of the number of signals connecting to each input

This is accomplished by modifying the routing cost function used by the neighbour expansion procedure in the directed search routing algorithm used to build the network. Although we do not wish to force nets, when encountering a $V_{routing}$ node that is part of a different connection, to connect to the same trace-pin, we do wish to make it preferential to do so in order to minimize the routing search space and hence runtime. By default, the routing cost function used for all nodes inside VPR is:

$$cost = back\_cost + this\_cost + astar \times expected\_cost \quad (1)$$

where *back_cost* is the congestion cost up to the current node, plus the *this_cost* of the node under consideration, and then the expected cost to the target scaled by an aggressiveness factor. Instead, for $V_{routing}$ nodes that are already part of another connection, we omit the *expected_cost* and discount *this_cost* by the occupancy of the new node, which indicates how many nets are already using it:

$$cost' = back\_cost + \frac{this\_cost}{node\_occupancy} \quad (2)$$

The intuition here is that the more nets that already pass through this node, the less likely it will be moved in subsequent routing iterations, and the more seriously the routing algorithm should consider it. A lower cost causes the preferred node to be removed from the heap much sooner than it would be otherwise, allowing the routing algorithm to follow the established connection to the trace-pin, yet does not force the router to take only this path. It must be noted that the new cost of a node must not take a value less than its predecessor (by discounting *back_cost* or using a negative value for *this_cost*) otherwise the tool will enter an infinite loop in which the cost of each node is endlessly reduced.

7.2 Debug-Time Matching

Given a designer-specified signal selection, we then process the text file describing the overlay network to build a custom bipartite graph containing only the desired signals, before applying the Hopcroft-Karp algorithm (as implemented in [4]) to find a maximum matching. A downstream tool can then be used to determine which signals to connect to which trace-pins, and thus compute the routing multiplexer bits required using a simple greedy algorithm. Subsequently, these bits can then be statically or dynamically reconfigured onto the FPGA.

In the absence of a large collection of realistic signal selections for each of our benchmark circuits, we have evaluated the feasibility of our work using random signal selections.

Circuit	Max Conn.	Comb&Seq. $\lvert V_{signals}\rvert$	Excl.	$\lvert V_{trace}\rvert$ Pins
`or1200`	15	3483	2	720
`mkDelayWorker32B`	5	7439	1	72
`stereovision1`	19	16211	11	3024
`stereovision0`	20	14937	12	3024
`LU8PEEng`	8	27657	3	1944
`stereovision2`	23	46646	-	11088
`bgm`	22	34966	66	7128
`LU32PEEng`	11	95026	1	7344
`mcml`	25	106555	4	17784

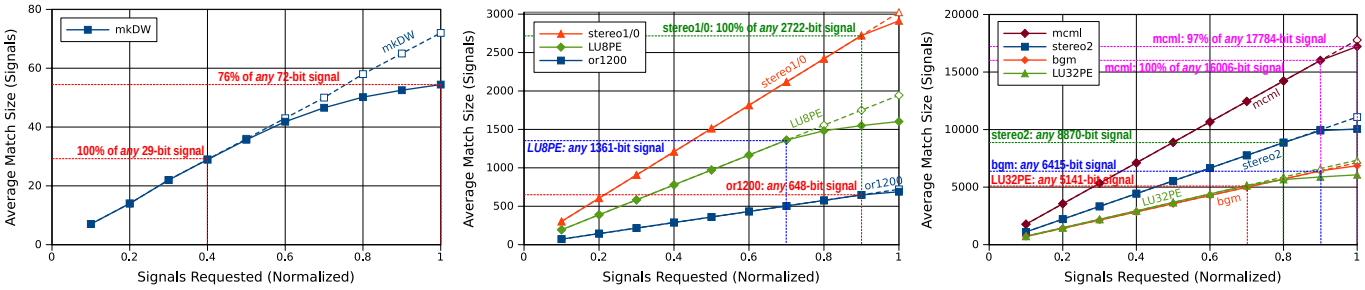
Table 3: Maximum network connectivity results

Although automated signal selection as proposed by [11, 10] may be used, these would only generate a handful of data-points. Instead, we would like to understand how the trace-buffer network fares for any signal that a designer may wish to select by using a sufficiently large sample size. For our experiments, we generated 100,000 signal selections randomly each at a different fraction of the trace-buffer network's capacity: from 0.1 to 1.0 in 0.1 increments. For example, if the trace-buffer network had a total of 720 input pins as for the `or1200` benchmark, then we randomly generated selections of 72 signals, 144, up to the full 720 for a total of 1,000,000 signal selections per circuit.

8. RESULTS

8.1 Maximum Network Connectivity

Table 3 shows the maximum network connectivity — the maximum number of trace-input trees that each signal belongs to, the number of circuit signals and trace-buffer inputs that exist for each circuit. For all but one of the nine benchmarks, not every internal signal could be incrementally connected to the trace-buffer network due to routing congestion. Upon further investigation, we found that only nets absorbed locally within a logic cluster suffered from this difficulty, caused by an inability to exit the cluster due to a lack of free resources in its vicinity. Unlike those nets that already had a presence on the global interconnect, for these local nets a new global route needed to be made from scratch using only the resources leftover from the original mapping. In very rare cases, this would be impossible. The number of signals that did fail in this manner are shown in the Excl. column, and represents at most 0.2% of all available combinational and sequential circuit signals. The number of trace-buffer inputs that exist for each circuit are also shown.



(a) Circuit: `mkDW` (b) Circuits: `or1200`, `stereo{1,0}`, `LU8PE` (c) Circuits: `stereo2`, `bgm`, `LU32PE`, `mcml`

Figure 9: Average match size — number of arbitrary signals that can be simultaneously forwarded by the overlay network to trace-buffers; dotted lines indicate the requested number of signals, solid lines indicate the average number of signals matched

As described earlier, this network connectivity parameter represents the guaranteed minimum number of signals for which a designer can observe *any* combination of, though in practice many more can be selected.

Figure 8a shows a histogram of the signal density at each trace-buffer input for the `mkDelayWorker32B` benchmark circuit, which contains only a single free memory-block to be reclaimed as a trace-buffer. In this circuit, all but five combinational and sequential signals can each be connected to 5 different trace-inputs. It would be expected that, on average, each input-pin would be connected to $\frac{7438\times5}{72} \approx 517$ signals, though in this particular instance 35 trace-pins exist which connect less than this value (with a minimum number of 5) leaving 37 pins which connect 517 or more signals (including a maximum of 1393 signals — almost 20% of all on-chip signals). A histogram for the largest circuit at our disposal, `mcml`, is produced in Fig. 8b, where it would be expected that each trace-input would be the target of approximately 126 signals. Here, a smaller proportion (40%) of trace-inputs are connected to by more than this value, indicating that there are some trace-buffers which are easier to access (i.e. more centrally located, as indicated in the histogram which shows a vertical scan-line ordering across the chip) than others, or that a tipping point is reached whereby it is cheaper for the routing tool to create new branches onto existing trees than to create entirely new trees.

If full observability into all circuits signals was strictly necessary, it may be possible to achieve this by increasing the channel width or the cluster output flexibility (Fc_out). We have observed that although increasing the channel width slack from $W_{min}+30\%$ to $+50\%$ had only a minimal effect on its network connectivity, in five of the nine benchmarks, all circuit signals can now be connected to the overlay network, whilst of the remaining four circuits, at most only 5 signals were impossible in the worst-case: `bgm`.

8.2 Average Match Size

Figure 9 shows the average match size returned by the maximum matching algorithm, where each data-point represents a sample size of 100,000 randomly-generated signal selections. This figure represents the average number of signals that can be simultaneously forwarded across the overlay network. The dotted lines of this graph show the number of signals requested by a designer, whilst the solid lines represent the average number of signals that can be forwarded across the network for observation. Where the lines coincide indicate

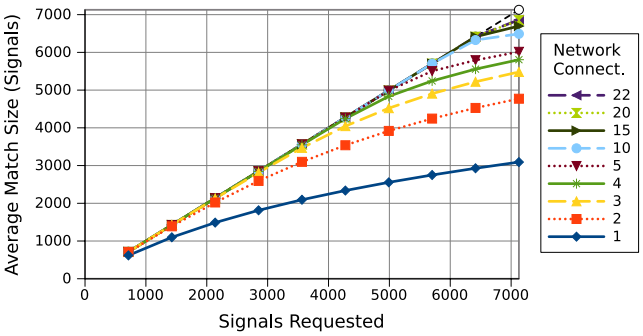
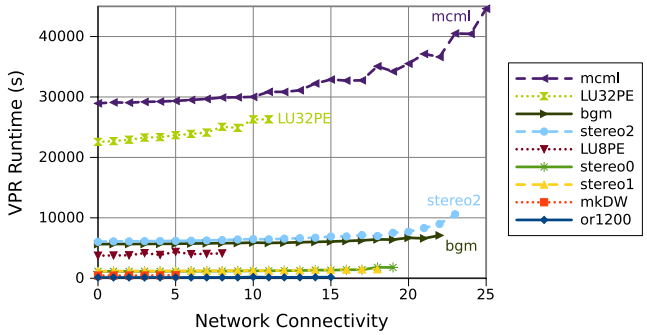


Figure 10: Network connectivity and match quality for `bgm`

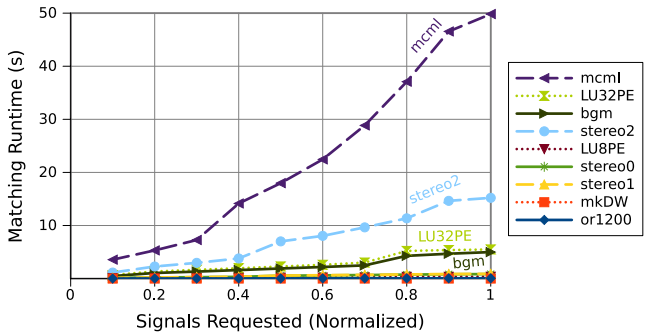
that a complete-match was made, where the lines diverge indicate that only partial-match was possible.

In Figure 9a, which corresponds to the `mkDelayWorker32B` benchmark, it can be seen that the probability of observing all of the desired signals decreases after approximately 40% of the network capacity — that is, after 29 signals are selected from a total capacity of 72. At full capacity, on average only 54 of the 72 signals requested can be matched. This is not a surprising result given the memory-constrained nature of this circuit, which contains only one free memory-block available for use as a trace-buffer. As stated in the previous section, the average number of signals that each trace-pin is expected to support for the `mkDelayWorker32B` circuit is 517 of the total 7439 signals; each time a trace-pin is used, 516 other signals are blocked from using this same pin, drastically reducing the flexibility of the overlay network. In contrast, the remaining non memory-limited circuits presented in Figures 9b and 9c show much more promising results: in most cases, the network can fully connect up to 80–90% of the trace-buffer capacity before conceding.

Figure 10 graphs how the number of signals observable through the overlay network varies with the network connectivity parameter, when applied to the `bgm` circuit. Intuitively, the more times that each signal is connected to a trace-buffer input, the less likely it will be blocked when a different signal is picked. However, these results show that it may not be necessary to connect each signal as many pins as possible — reducing this network connectivity parameter to 10 or 15 has no effect on the signals observable when requesting 90% trace capacity, and only a 2 or 5% reduction at 100% capacity when compared with the maximum connectivity value of 22.



(a) Compile-time: total VPR runtime for overlay network insertion at various connectivities; connectivity=0 represents baseline with no network or instrumentation



(b) Debug-time: maximum matching runtime, per selection

Figure 11: Runtime overhead

8.3 Runtime

Figure 11a shows the total VPR runtime for building the overlay network, for each connectivity parameter possible, averaged over 10 tries. An X value of zero indicates the baseline measurement, which does not include any trace-buffers nor an overlay network. X values greater than zero specify the total runtime for the standard CAD stages: packing, placement, routing, as well as the additional stage of incremental-routing to embed our overlay network. The difference in runtime represents the additional overhead of our overlay network; on average this is a 34% increase on the baseline, and in the worst-case this reaches 76% for stereovision2. As expected, runtime increases with the network connectivity value, with the gradient increasing more rapidly towards the tail-end of each circuit as it reaches its breaking point. However, it may not be necessary to push each circuit to this point as Figure 10 from the previous subsection showed. We anticipate that with greater focus on optimizing the CAD algorithms, we can further reduce this overhead.

The runtime for finding a maximum signal to trace-pin match is charted in Figure 11b — this is the average time required to recompute a matching network assignment to support new signal selections. In the worst case, for the largest `mcml` benchmark where the full trace-buffer capacity is requested, a solution can be returned in less than 50 seconds, with the relationship between runtime and the number of signals requested appearing to be linear. This contrasts with the time required to either fully or incrementally recompile the circuit to create a new point-to-point trace-buffer configuration: in the previous figure we observed 30,000 seconds to fully compile an uninstrumented instance of `mcml`, whilst reference [8] stated that approximately 2,000 seconds was required to incrementally utilize 75% of the on-chip trace capacity. Matching runtime can be improved further by implementing our techniques in a more efficient programming language instead of Python.

Circuit	Original T_cpd (ns)	Instrum'ed T_cpd (ns)	Change
`or1200`	21.6	21.9	+1.4%
`mkDelayWorker32B`	7.4	7.6	+2.7%
`stereovision1`	5.1	6.1	+19.6%
`stereovision0`	4.1	5.8	+41.4%
`LU8PEEng`	134.0	136.0	+1.5%
`stereovision2`	15.0	16.6	+10.7%
`bgm`	25.8	27.1	+5.0%
`LU32PEEng`	134.2	137.4	+2.4%
`mcml`	96.7	98.7	+2.1%
Geomean	23.6	25.7	+9.0%

Table 4: Effect of overlay network on critical-path delay

8.4 Circuit Delay

A comparison of the critical-path delay before and after inserting our virtual overlay network on each of our benchmark circuits is shown in Table 4. Currently, because our CAD algorithms are routability-driven rather than timing-driven, on average the network incurs a 9.0% penalty to the critical delay, with a worst-case of 41.4% for `stereovision0` which has the shortest critical-path. We believe that these results may be a little on the conservative side due to the nature of the circuits and CAD tools involved, where the majority of the critical-path delay — between 53% and 89% (geomean at 72%) — is made up of logic delay rather than routing delay.

Given that we add our overlay network incrementally, that is, only after the original user circuit is fully-compiled, the critical-path delay of the newly instrumented design is due entirely to the connections added by this network. If the observability that the trace infrastructure provides is not required, the circuit can revert to operating at its original, uninstrumented, clock frequency. During prototyping, however, it is unlikely that circuits will be operated at this maximum frequency, perhaps limited by off-chip (inter-FPGA) communication and hence timing degradation may not be a critical issue. Despite this, one promising direction for future work is to apply pipelining techniques to the overlay network in order to reduce its effect on delay — a technique particularly relevant for this application because any increase in signal latency will not affect its observability.

9. CONCLUSION

FPGAs are increasingly being used as prototyping platforms. Compared to software simulators, these prototypes can achieve significantly higher operating frequencies allowing designers to increase their verification coverage by several orders of magnitude. When unexpected or erroneous circuit behaviour is detected, designers begin a debugging procedure so that they may understand the root cause of their anomaly. However, the key challenge with debugging FPGA prototypes is their lack of built-in observability; unlike simulation, designers cannot simply probe any signal of their choice. One common solution to this problem is to insert trace-buffer instrumentation; these blocks seek to record a

small, predetermined subset of signals into on-chip memory for subsequent analysis.

Existing academic work [8, 6], and many of the current commercial offerings [24, 1] require a designer to preselect the signals they wish to observe at compile-time, after which point-to-point connections are made for each signal to a trace-buffer. In this work, we have proposed a method which aims to allow designers to look at *any* subset of combinational or sequential signals in their circuit at debug-time, relieving them of the need to predetermine a selection beforehand. Due to on-chip memory constraints, it is not possible to make a dedicated trace connection for each signal; hence, we pass each signal through an overlay network which multiplexes these connections between the available trace-buffers, thus allowing this signal selection to be deferred to debug-time. Unlike a similar approach adopted in [19], we do not use soft-logic for this purpose, opting instead to utilize switch- and connection-box routing multiplexers that form part of the FPGA fabric. Because we reclaim these routing multiplexers from those that were leftover in the original circuit mapping, the area overhead of our work is essentially zero.

Due to routing requirements, it would be impractical to build a fully-populated crossbar network in which all inputs can be forwarded to all outputs; hence we build a blocking network in which only a reduced amount of connectivity exists. To decide which of the input signals to connect to the output trace-pins, we apply a maximum matching algorithm to the bipartite graph that represents our network to find the optimal solution with the highest number of observed signals. Once this assignment has been determined, the configuration memory of those routing multiplexers are reconfigured using either static or dynamic techniques. Our experiments have shown that for the majority of the benchmark circuits that were investigated, we were able to build an overlay network connecting to over 99.8% of all circuits signals whilst increasing initial CAD runtime by an average of 34%. However, once this network has been built, it can be reconfigured as many times as necessary to forward any set of signals through the overlay network to approximately 80–90% of the on-chip trace capacity, in no more than 50 seconds.

Acknowledgements

The authors are grateful to Altera for supporting this work, and would also like to acknowledge Wayne Luk and the Department of Computing at Imperial College London, where most of this research was conducted over the summer of 2012.

10. REFERENCES

[1] Altera. Quartus II Handbook Version 12.0 Volume 3: Verification. http://www.altera.com/literature/hb/qts/qts_qii5v3.pdf, June 2012.

[2] S. Asaad, R. Bellofatto, B. Brezzo, C. Haymes, M. Kapur, B. Parker, T. Roewer, P. Saha, T. Takken, and J. Tierno. A Cycle-Accurate, Cycle-Reproducible Multi-FPGA System for Accelerating Multi-core Processor Simulation. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '12, pages 153–162, 2012.

[3] M. Bourgeault. Altera's Partial Reconfiguration Flow. http://www.eecg.utoronto.ca/~jayar/FPGAseminar/FPGA_Bourgeault_June23_2011.pdf, June 2011.

[4] D. Eppstein. Hopcroft-Karp Bipartite Max-Cardinality Matching and Max Independent Set (Python Recipe). http://code.activestate.com/recipes/123641-hopcroft-karp-bipartite-matching/, April 2002.

[5] H. Foster. Challenges of Design and Verification in the SoC Era. http://testandverification.com/files/DVConference2011/2_Harry_Foster.pdf.

[6] P. Graham, B. Nelson, and B. Hutchings. Instrumenting Bitstreams for Debugging FPGA Circuits. In *Field-Programmable Custom Computing Machines, FCCM'01. The 9th Annual IEEE Symp. on*, pages 41–50, March 2001.

[7] E. Hung and S. J. E. Wilton. Speculative Debug Insertion for FPGAs. In *FPL 2011, International Conference on Field Programmable Logic and Applications; Chania, Greece*, pages 524–531, September 2011.

[8] E. Hung and S. J. E. Wilton. Limitations of Incremental Signal-Tracing for FPGA Debug. In *FPL 2012, International Conference on Field Programmable Logic and Applications; Oslo, Norway*, pages 49–56, August 2012.

[9] Y. S. Iskander, C. D. Patterson, and S. D. Craven. Improved Abstractions and Turnaround Time for FPGA Design Validation and Debug. In *FPL'11, Proceedings of the 2011 21st International Conference on Field Programmable Logic and Applications*, pages 518–523, 2011.

[10] H. F. Ko and N. Nicolici. Algorithms for State Restoration and Trace-Signal Selection for Data Acquisition in Silicon Debug. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(2):285–297, 2009.

[11] X. Liu and Q. Xu. On Signal Selection for Visibility Enhancement in Trace-Based Post-Silicon Validation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 31(8):1263–1274, Aug. 2012.

[12] L. McMurchie and C. Ebeling. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs. In *Proceedings of the 1995 ACM Third Int'l Symp. on Field-Programmable Gate Arrays*, FPGA '95, pages 111–117, 1995.

[13] Y. O. M. Moctar, N. George, H. Parandeh-Afshar, P. Ienne, G. G. Lemieux, and P. Brisk. Reducing the Cost of Floating-Point Mantissa Alignment and Normalization in FPGAs. In *Proceedings of the 20th ACM/SIGDA Int'l Symp. on Field-Programmable Gate Arrays*, FPGA '12, pages 255–264, 2012.

[14] B. Quinton and S. Wilton. Concentrator Access Networks for Programmable Logic Cores on SoCs. In *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pages 45–48 Vol. 1, May 2005.

[15] J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. Anderson. The VTR Project: Architecture and CAD for FPGAs from Verilog to Routing. In *Proceedings of the 20th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA'12, pages 77–86, February 2012.

[16] N. Shah and J. Rose. On the Difficulty of Pin-to-Wire Routing in FPGAs. In *FPL 2012, International Conference on Field Programmable Logic and Applications; Oslo, Norway*, pages 83–90, August 2012.

[17] Synopsys. Identify: Simulator-like Visibility into Hardware Debug. http://www.synopsys.com/Tools/Implementation/FPGAImplementation/CapsuleModule/identify_ds.pdf, Aug. 2010.

[18] S. Teig. Programmable logic devices in 2032? (FPGA2012 Pre-Conference Workshop). http://tcfpga.org/fpga2012/SteveTeig.pdf, February 2012.

[19] Tektronix. Certus Debug Suite. http://www.tek.com/sites/tek.com/files/media/media/resources/Certus_Debug_Suite_Datasheet_54W-28030-1_4.pdf, July 2012.

[20] E. Vansteenkiste, K. Bruneel, and D. Stroobandt. Maximizing the Reuse of Routing Resources in a Reconfiguration-Aware Connection Router. In *FPL 2012, International Conference on Field Programmable Logic and Applications; Oslo, Norway*, August 2012.

[21] T. Wheeler, P. Graham, B. E. Nelson, and B. Hutchings. Using Design-Level Scan to Improve FPGA Design Observability and Controllability for Functional Verification. In *FPL '01: Proceedings of the 11th International Conference on Field-Programmable Logic and Applications*, pages 483–492, 2001.

[22] M. Wirthlin, B. Nelson, B. Hutchings, P. Athanas, and S. Bohner. FPGA Design Productivity: Existing Limitations and Root Causes. http://www.chrec.org/ftsw/FDP_Session1_Posted.pdf, June 2008.

[23] Xilinx. Virtex-6 FPGA Memory Resources: User Guide (UG363 v1.6). http://www.xilinx.com/support/documentation/user_guides/ug363.pdf, April 2011.

[24] Xilinx. ChipScope Pro Software and Cores, User Guide. http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_2/chipscope_pro_sw_cores_ug029.pdf, July 2012.

[25] Xilinx. Partial Reconfiguration of Xilinx FPGAs Using ISE Design Suite (WP374 v1.2). http://www.xilinx.com/support/documentation/white_papers/wp374_Partial_Reconfig_Xilinx_FPGAs.pdf, May 2012.